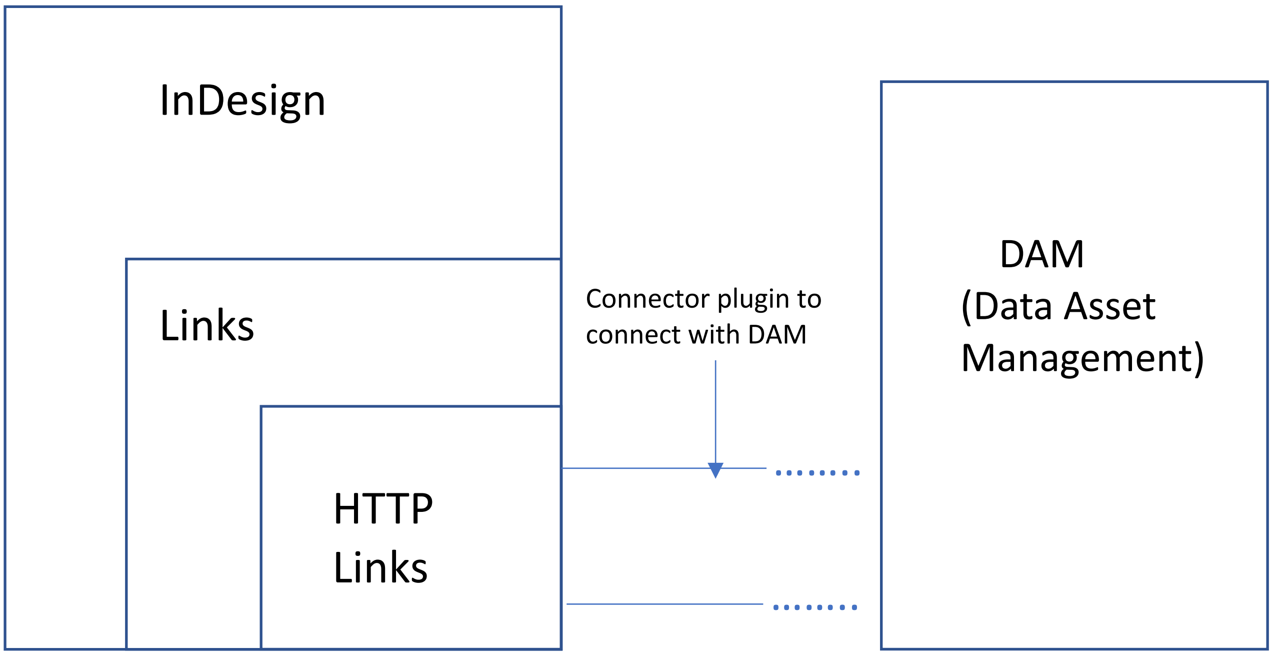
Custom HTTP Links SDK Sample

Background

This SDK is intended to describe how write connector for DAM leveraging InDesign link architecture. Following picture describes relationship between different blocks:


InDesign
Links
HTTP Links
Connector plugin to connect with DAM
DAM (Data Asset Management)

Prerequisites

1. **Links Architecture**: Refer to Chapter 6 : **Links** of plugin-programming-guide-vol2.pdf.
2. **Definition of URI and scheme** (refer to "Support Your on links" heading on Page 130 of plugin-programming-guide-vol2.pdf.)
3. **Service Provider**: (interface IK2ServiceProvider) — The entity that identifies the object as a responder and indicates the events of interest. (Refer heading "Responders" in Chapter 2 of plugin-programming-guide-vol1.pdf)

Contents

There are 3 parts to CustomHttpLinks SDK sample:

1. CustomHttpLinkUI
2. CustomHttpLink
3. Localserver

See CusHttpLnkSampleScript.jsx in this plug-in's examplefiles folder for sample JavaScript. This script requires a native plugin (in this case, CustomHttpLink) to be installed which handles the HTTP Links of given scheme.

1. CustomHttpLinkUI

This is a UI plugin. The plugin has a dropdown populated with assets: asset1.jpg, asset2.jpg, …, asset10.jpg. The actual assets are inside the “localserver/image” directory. The low resolution version of these assets are placed in “localserver/image/fpo” directory.

In the "/image/" directory, there are some royalty free images for a headstart, from https://www.pexels.com, and their license states - https://www.pexels.com/photo-license/.
DISCLAIMER: Adobe doesn't recommend using these images outside this sdk sample. Please refer to Adobe Image Use Rights - https://www.adobe.com/in/legal/permissions/image-notice.html.

If you want to use your own assets, add them with name "asset1.jpg", "asset2.jpg", "asset3.jpg", ..., "asset10.jpg" to the "/image/" directory.
Add their corresponding low resolution versions with same name to the "/image/fpo/" directory.
For a given sample asset:
 * Original URI format: http://localhost:3000/image/abc.jpeg
 * FPO URI format: http://localhost:3000/image/abc.jpeg/fpo

Following tasks can be performed using this plugin:
- Login or Logout from the DAM (localserver in this case) from the flyout menu
- Place the assets using “Place” button

2. CustomHttpLink
This is a model plugin to extend the InDesign’s URL linking capability to a custom DAM.

3. Localserver
This is a sample nodejs server (refer to localserver/index.js) . Nodejs based server is provided to acts as DAM. It provides basic API’s like:
- Get the high res and low res asset
- Get asset information APIs (single and batch)

Run the Sample
This SDK sample is for reference to write connector easily to specific DAM/CMS based on InDesign's URL linking capability.
In order to try this sample, you need to run the sample's local server(based on nodejs) with the steps given below -
1. Navigate to localserver directory
2. Install nodejs from their website (https://nodejs.org/en/)
3. To install the required packages, run following command:
```
$ npm install
```
4. Run following command to start the server
```
$ node index.js
```
5. If everything is working, console should display "server is listening on 3000"
6. After running the local server, build and play around with the sdk sample native code of CustomHttpLink and CustomHttpLinkUI.

7. To connect with the local server, select 'Login' option from the CustomHttpLinkUI panel flyout menu.

Details on writing custom connector

To create custom plugins to handle http calls, first a scheme needs to be implemented. Since http scheme can be used by many plugins, so it is advised to use custom scheme. Benefit of using scheme based system is that it is not dependent on server URI/address. So its easy to distribute plugin. Scheme name should be used wisely so that it doesn't end up clashing with another plugin. In such cases plugin which registers scheme first, will be given priority, also assert will be shown in those case.

1. Implement IHTTPLinkSubsystemObjectFactory interface on the http link resource connection provider boss that is registered with the service registry as a kHTTPLinkResourceConnectionProvider. This is mandatory step. Look into CustomHttpLink.fr for reference:

```
Class
{
      kCusHttpLnkLinkResourceProviderBoss,
      kInvalidClass,
      {
            IID_IK2SERVICEPROVIDER,
kHTTPLinkSubsystemObjectProviderImpl,
            IID_IHTTPLINKSUBSYSTEMOBJECTFACTORY,
kCusHttpLnkLinkResourceFactoryImpl,
      }
},
```

Implementation should return list of schemes plugin supports. For this example we registering "myhost" scheme. It creates objects when query functions are used. It return four objects : handler, stateupdater, connection and wrapper object. Custom impplementations can be return from these functions.
MANDATORY INTERFACES that needs to implemented are : IID_IHTTPLINKRESOURCESERVERAPIWRAPPER, IID_IHTTPLINKRESOURCECONNECTION. For others default implementation can be used. They are discussed in details below.

2. The are useful boss objects and interfaces which are related to http links implementation:

   i) **kHTTPAssetLinkResourceServerHelperBoss**: This include following interfaces:

      a. IID_IHTTPLINKRESOURCESERVERAPIWRAPPER - Mandatory to implement.

IHTTPLinkResourceServerAPIWrapper deals with high level server API/functionalities. Some of its responsibilities include fetching metadata, checking server connection , generation of session token etc. This is mandatory implementation. To subclass create a boss based on kHTTPAssetLinkResourceServerHelperBoss with the implementation of this interface. And return this connection instance from above implementation of IHTTPLinkSubsystemObjectFactory. Refer CusHttpLnkResourceServerAPIWrapper.cpp and CusHttpLnkLinkResourceFactory.cpp.

b. IID_IHTTPLINKRESOURCESERVERCACHE - This interface provides deals with caching mechanism for files and metadata for HTTP links. This interface is responsible for storing asset metadata. Ideally it should persist across the session. Also it should clear cache and metadata also, so that it doesn't bloat. To subclass create a boss based on kHTTPAssetLinkResourceServerHelperBoss with the implementation of this interface. And return this connection instance from the IHTTPLinkSubsystemObjectFactory.

c. IID_IHTTPASSETDOWNLOADMANAGER - IHTTPAssetDownloadManager manages downloads for http link assets (see IHTTPLinkSubsystemObjectFactory.h for more details). It is called when specific asset is not found in cache from IHTTPLinkResourceServerCache. Include this interface on http asset link resource server helper boss which is registered as kHTTPAssetLinkResourceServerHelperBoss. For custom implementation aggregate this interface with IID_IHTTPLINKRESOURCESERVERAPIWRAPPER on kHTTPAssetLinkResourceServerHelperBoss.

d. IID_IHTTPLINKASSETURIPREPROCESSOR - This interface is experimental and for internal use only.

ii) kHTTPAssetLinkResourceHelperBoss:

a. IID_IHTTPLINKRESOURCECONNECTION - Mandatory to implement. IHTTPLinkResourceConnection deals with connection details of the server. It contains function related to connection with server. Use this interface to implement logic related to login details, passing session token etc.

b. IID_IHTTPLINKRESOURCESTATEBATCHUPDATER - IHTTPLinkResourceStateBatchUpdater contains functions for updating states in batch mode. This will only be used when web-api supports batch mode. For custom implementation return true from SupportsHTTPResourceStateBatchUpdater function of IHTTPLinkResourceServerAPIWrapper. Then subclass this interface by creating a boss based on kHTTPAssetLinkResourceHelperBoss with the implementation of this interface.

iii) kHTTPAssetLinkResourceStateUpdaterBoss:

a. IID_ILINKRESOURCESTATEUPDATER - If plugin needs to change how metadata is handled or state updation logic. This interface can be implemented to alter it. Look into kCusHttpLnkLinkResourceStateUpdaterBoss in CustomHttpLink.fr. Currently it is stub implementation, functions can be overridden to alter implementations. Derive from HTTPAssetLinkResourceStateUpdater to alter its functionalities.

iv) kHTTPAssetLinkResourceHandlerBoss:

a. IID_ILINKRESOURCEHANDLER : To change behaviour of HTTPAssetLinkResourceHandler. Implement this interface and derive from HTTPAssetLinkResourceHandler. One such interface is showcased in CusHttpLnkLinkResourceHandler.cpp. If you want different resource to be displayed in UI and different name for internal use, GetLongResourceName can be overriden. Refer to publiclib/links/HTTPAssetLinkResourceHandler.cpp

b. IID_ILINKRESOURCEHELPERHANDLER : In CusHttpLnkLinkResourceHelperHandler, ReplaceWithOriginalAsset is used to perform action (kReplaceFPOWithOriginalLinkActionID).